Deep Reinforcement Learning for Derivative Pricing and Trading Strategy

***Abstract*—With significant improvements in deep learning techniques, computational power and the amount of data available, efficient and effective derivatives pricing is possible. In this paper, we try to apply deep reinforcement learning using recurrent neural network architecture for pricing derivatives. This approach is an enhancement over traditional fair pricing models which are efficient only to price standard derivatives, as they assume a certain underlying distribution for underlying. But Reinforcement learning models can price derivatives directly from the underlying data without assuming any parametric form; hence, certain problems like jump discontinuities and fat-tails that occur in the real world derivatives prices can be avoided using deep reinforcement learning and Recurrent Neural Networks.**

Keywords - Deep Reinforcement Learning, Recurrent Neural Networks, Derivatives Pricing, Long short-term memory

I. INTRODUCTION

Financial markets follow a complex pattern and are characterized by a stochastic (time interchanging) behaviour resulting in multivariate and highly nonlinear option pricing functions. Parametric models describe a stationary nonlinear relationship between a theoretical option price and various variables. There is also evidence indicating that market participants change their option pricing attitudes from time to time. Parametric Option Pricing Models (OPMs) may fail to adjust to such rapidly changing market behaviour. That is why efforts are being made to develop nonparametric techniques that can overcome the limitations of parametric OPMs. In addition to this, market participants always have a need for more accurate OPMs that can be utilized in real-world applications [1].

In this paper, we applied reinforcement learning using the recurrent neural network for option pricing and determining the trading strategy for options. While LSTM has been known to be immensely good for time-series modeling of derivatives which follow highly non-linear and complex patterns, recommending trading strategies along with predicting prices can make a model more useful. As such, we wish to test whether reinforcement learning applied to traditional RNN has potential to improve performance of both options pricing and recommending accurate option strategies.

II. RELATED WORK

In this section, we present the overview of major techniques that are relevant to our work.

A. Reinforcement Learning (RL)

Reinforcement Learning is an important type of Machine Learning where an agent learns how to interact with the environment to maximize the expected present value of cumulative rewards. The reward function is a domain specific function. In case of derivative pricing, the reward for a certain state-action pair will simply be the payoff obtained by the trader on executing the action at that particular state. This means reward for a particular action can vary based on the state the trader is in.

Let R (s_t, a_t) be the state-action reward function for action, a_t taken at state, s_t to move to state, s_{t+1}. Then, the reward function i.e. the sum of expected future rewards of a given strategy (or policy), π discounted to time, t can be written as:

$$V^{\pi}(s_t) = E[\sum_{\tau=t}^{T} e^{-r(\tau-t)} R(s_\tau, \pi(s_\tau)) \mid s_t] \quad (1)$$

B. Sequence to Sequence Long Short-Term Memory (LSTM)

Deep Neural Networks (DNNs) are the Artificial Neural Networks with multiple hidden layers between input and output layers. DNNs can model complex non-linear relationships. In order to capture the context in the sentences, we use Recurrent Neural Networks (RNNs), a very prominent class of DNN. RNN architecture is well suited for learning from the sequence of values in a time-series dataset, to solve the problem of predicting the outputs in the next steps based on history of previous inputs. LSTM units are a class of RNNs that can capture both long term and short term dependencies and they are relatively immune to problems of vanishing or exploding gradients even with 1000 discrete time steps. Hence Recurrent Neural Networks like LSTM have become indispensable in building any type of sequential models [2].

III. DATASET PREPARATION

A. *Options and Underlying Data*

The options and underlying data are taken from the source, http://tradecatcher.blogspot.com/p/options-ieod.html, which are publicly available. This data are for European Call options on *Nifty* index from 13th March 2018 to 26th April 2018 with one minute frequency interval. The maturity date of all the selected options is 26th April 2018 and their strike-price *K* is Rs.10,000. Call option data has the following columns: *DateTime*, *Open* - opening option price for the interval, *High* - highest option price for the interval, *Low* - lowest option price for the interval, *Close* - closing option price for the interval and *Volume* - volume of

options traded in the interval as given in Table I. The high trading volume shows high option liquidity. Nifty index data has the following columns: *DateTime*, *Open_Nifty* - Opening index price for the interval, *High_Nifty* - highest index price for the interval, *Low_Nifty* - lowest index price for the interval, *Close_Nifty* - closing index price for the interval and *Volume_Nifty* - volume of the index traded in the interval as given in Table II.

TABLE I. SNAPSHOT OF CALL OPTION DATASET

DateTime	Open/Rs	High/Rs	Low/Rs	Close/Rs	Volume
3/13/2018 9:17	515.1	515.1	515.05	515.05	150
3/13/2018 9:18	519.95	519.95	519.95	519.95	75
3/13/2018 9:19	524.9	524.9	524.9	524.9	75
3/13/2018 9:20	525	525	525	525	75

TABLE II. SNAPSHOT OF NIFTY DATASET

DateTime	Open_ Nifty/Rs	High_ Nifty/Rs	Low_ Nifty/Rs	Close_ Nifty/Rs	Volume_ Nifty
3/13/2018 9:17	10397.3	10413.2	10397.3	10412	55
3/13/2018 9:18	10412.2	10415.3	10409.1	10414.8	56
3/13/2018 9:19	10415.8	10427.8	10415.8	10427	58
3/13/2018 9:20	10427.3	10432.6	10427.3	10429.8	57

B. Labelled Dataset for training

The original dataset does not contain the actions that were actually taken by an option trader in certain point of time. We considered four possible actions for the option trader. The actions are "Buy" when a trader takes long position, "Sell" when a trader takes short position, "No-Hold" when a trader already sold the option and does not hold the option any longer and "Hold" when a trader holds the option and does not sell it.

We use momentum trading strategy in order to incorporate actions in the training dataset. We assumed that the right actions at each instance of time were those which a rational trader would have performed at that point of time based on the 15x30 minute Exponential Moving Average (EMA) crossover system as discussed in [3]. The models are then trained based on the closing option prices for the interval and the actions derived from this technical analysis.

C. Feature Engineering

We considered seven features for training our models for options pricing and trading strategy. These are : Ratio of opening index price to option strike price (*Open_Nifty*/*K)* , Ratio of closing index price to option strike price (*Close_Nifty*/*K)* , Ratio of highest index price to option strike price (*High_Nifty*/*K*), Ratio of lowest index price to option strike price (*Low_Nifty*/*K*) , Trading Position (1 if the trader holds an option for the interval of time and 0 otherwise), Time to Expiry (in days) given by number of days between the trade day and the expiry date of the option and Volume of stocks traded in the exchange at each instant of time (Volume_Nifty). These seven features collectively define the market state s_t at a certain point of time for our model.

D. Data Preprocessing

We scale the continuous data using Robust Scaler package available in the *sklearn* library of Python to limit the range of the input and output values as they are scaled to their respective interquartile ranges. This makes the data less sensitive to outliers and substantial changes in trends in the trading data. For the actions data, we use one-hot encoding.

IV. METHODOLOGY

We predict the option prices and form trading strategy by applying reinforcement learning using sequence to sequence LSTM. We use a sliding window of 50 time steps as input to the model and predict the option prices and actions for the next 5 time-steps (each time-step being for one minute). The implementation of the model is done using *Keras* wrapper which runs on *Tensorflow* available in Python. Table III shows the architecture used for the model. Hidden units for the LSTM are tuned using the validation data. Dropout is used to avoid overfitting and improve performance of the model. The Repeat Vector repeats the output from Dropout-1 so that each of the next time steps will get the same input but a different hidden state. The two TimeDistributed layers are used to get the option prices and actions for next 5 time-steps. The Concatenate layer combines the prices and actions into a single vector.

TABLE III. MODEL SUMMARY

Layer	Output Shape	Connected to
InputLayer	(None, 50 ,7)[a]	-
LSTM-1	(None, 40)[b]	InputLayer
Dropout-1	(None ,40)	LSTM-1
RepeatVector	(None, 5 , 40)	Dropout-1
LSTM-2	(None, 5, 40)[c]	RepeatVector
Dropout-2	(None, 5, 40)	LSTM-2
TimeDistributed Dense -1	(None, 5, 1)[d]	Dropout-2
TimeDistributed Dense -2	(None , 5 , 4)[e]	Dropout-2
Concatenate	(None , 5, 5)	TimeDistributed Dense-1, TimeDistributed Dense -2

[a] 50 time steps input data with 7 features to the model

[b] 40 hidden units for LSTM 1 found using validation data

[c] 40 hidden units for LSTM 2 found using validation data

[d] Layer for getting the prices 5 time steps ahead using linear activation function

[e] Layer for getting the actions 5 time steps ahead using softmax activation

RL is incorporated in the traditional LSTM network by using custom loss function based on reward function as sum of the present value of expected future rewards to train the model.The rewards are different for different actions taken by the option trader. For "Buy" action when option is priced at V, trader will incur a negative reward –V along with a

transaction cost which is usually around 0.2% of the transaction amount and hence, we consider the reward during the buy action to be -1.002V. A similar transaction cost will be incurred during "Sell" action and hence the reward in such case will be (V – 0.002V) = 0.998V. For "no-hold" action, the reward is taken to be 0. For "hold" action, there will be a small negative reward equivalent to the opportunity cost of holding the option which is considered to be around -0.01V.

The goal of our RL algorithm is to find out set of actions ad option prices for each of the one-minute consecutive time-steps up to option expiry time so as to maximize the reward function. We are however, predicting the option prices and action for 5 time-steps. So, the future rewards are taken only for next 5 time-steps for defining the reward function. The discount rates (usually less than 10% annually) will be insignificant for 5 time-steps and hence, we consider the reward function as the cumulative sum of these future rewards without discounting.

Let V(i) be the actual option prices and V'(i) be the option prices predicted by the neural network model for time-step i where i=1,2,3,....n. We use similar notations for the actual actions a(i) and predicted actions a'(i) and the corresponding actual rewards r(i) and predicted rewards r'(i). The loss function taken for RL is given by:

$$L_1 = [\sum_{i=1}^{n} (r(i) - r'(i))^2] \quad (2)$$

We can expect that minimizing this loss function for training dataset over the next "n" time-steps, here 5 time-steps, can result in selecting the optimal policy of actions based on the predicted prices through the reinforcement learning approach.

The prices and actions can also be predicted by using traditional LSTM loss functions. The loss functions $L_2$and L_3 are associated with categorical accuracy of action prediction and the accuracy of price prediction respectively and are given by:

$$L_2 = [\sum_{i=1}^{n} a(i) * \log(a'(i)) * w(a'(i))] \quad (3)$$

$$L_3 = [\sum_{i=1}^{n} V(i) - \sum_{i=1}^{n} V'(i)]^2 \quad (4)$$

The loss L_2 is called as the weighted categorical cross-entropy loss function. This is slightly modified version of original categorical entropy function defined in most software packages, by multiplying with the weight term w(a'(i)). The weight term is used to give unequal weightages to losses associated with predictions of different actions. This is done to handle the class imbalance for the actions in the training data.

We compare the performance of the traditional LSTM model using L_2 and L_3 losses, RL based LSTM model using L_1 loss function and a hybrid model of the two approaches using combination of L_1, L_2 and L_3 losses. This helps us analyse how reinforcement learning can improve the performance of derivative price prediction and actions prediction when added to traditional LSTM model.

V. RESULTS AND DISCUSSION

We have measured both categorical accuracy and weighted categorical accuracy for analyzing the performance of our model. In weighted categorical accuracy, we assign weights to the accuracy measures of our actions. In this case, just like weighted loss, we have considered weights [30,1,1,30] for weighted categorical measure, i.e. "Buy":30, "No-Hold":1, "Hold":1, "Sell":30. This gives higher weightage to the accuracies of buy and sell actions relative to the hold and no-hold which could be important to address the issue of class imbalance here as buy and sell actions occur nearly 30 times less in our dataset as compared to the other two actions.

In Table V, we present the parameters used for tuning the model to obtain best validation results (the first row and dropout = 0.2) and in Table VI, we use the tuned model to compare performance with the test results. Validation split ratio is 10% of trained dataset and test-split ratio is 15% of overall dataset.

TABLE V. VALIDATION FOR HYPER-PARAMETERS

Epoch	Batch Size	Hidden Units 1	Hidden Units 2	R2 Score	Weighted Categorical Accuracy
10	30	40	40	0.983	0.684
10	30	45	45	0.977	0.569
15	30	40	40	0.976	0.645
10	45	40	40	0.976	0.650

TABLE VI. TEST RESULTS

Model	RL + RNN	RNN only
R2 score	0.985	0.981
Weighted Categorical Accuracy	0.704	0.664
Categorical Accuracy	0.679	0.645

VI. RECOMMENDATIONS

Our model considered only one strike price for the training, validation and test datasets. However, options for different strike prices can be considered for training the model deeper to improve the accuracy of the prices and especially the action predictions as we feel that strike price is indeed a significant factor. This may have the potential to increase the pricing and strategy prediction accuracy of reinforcement learning with RNN further.

VII. CONCLUSIONS

From our results, we conclude that incorporating reinforcement learning [4] with LSTM provides the best prediction accuracy of trading strategies compared to using either only LSTM or only reinforcement learning. However, we also find that prediction accuracy of option prices is not really significantly different for model using LSTM based loss function alone and the model incorporating RL based loss function alongwith LSTM loss function. This means that reinforcement learning doesn't really substantially improve the pricing prediction performance though it is improving the categorical accuracy of predicting option strategies. .However, since predicting trading strategies is an important part of predictive modeling of any derivative trading, we can conclude that including reinforcement learning algorithm together with RNN or LSTM has potential to improve the performance of predictive models of derivative trading.

VIII. REFERENCES

[1] Amit Deoda, "Option Pricing using Machine Learning techniques" - Indian Institute of Technology Bombay - June 2011, submitted in partial fulfilment of the requirements for the degree of Bachelor of Technology And Master of Technology, unpublished

[2] Mohamed Akram Zaytar, Chaker El Amrani, "Sequence to Sequence Weather Forecasting with Long Short-Term Memory Recurrent Neural Networks", International Journal of Computer Applications (0975 - 8887), Volume 143 - No.11, June 2016

[3] Puchong Praekhaow "Determination of Trading Points using the Moving Average Methods", International Conference for a Sustational Greater Mekong subregion, Volume: GMSTEC, June 2010

[4] Thomas Grassl, "A reinforcement learning approach for pricing derivatives", 2010, unpublished